Introduction to Tree (Part 2)

Spanning Tree, Tree Traversal (Suplementary), and Some Applications of Tree (Suplementary)

MZI

School of Computing
Telkom University

SoC Tel-U

May 2023

Acknowledgements

This slide is composed based on the following materials:

1. *Discrete Mathematics and Its Applications*, 8th Edition, 2019, by K. H. Rosen (main).
2. *Discrete Mathematics with Applications*, 4th Edition, 2018, by S. S. Epp.
3. *Mathematics for Computer Science*. MIT, 2010, by E. Lehman, F. T. Leighton, A. R. Meyer.
4. Slide for Matematika Diskret 2 (2012). Fasilkom UI, by B. H. Widjaja.
5. Slide for Matematika Diskret 2 at Fasilkom UI by Team of Lecturers.
6. Slide for Matematika Diskret. Telkom University, by B. Purnama.

Some of the pictures are taken from the above resources. This slide is intended for academic purpose at FIF Telkom University. If you have any suggestions/comments/questions related with the material on this slide, send an email to `<pleasedontspam>@telkomuniversity.ac.id`.

Contents

Contents

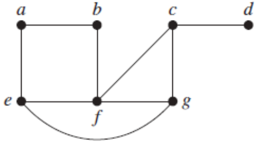
Spanning Tree

Spanning Tree

A spanning tree of a graph G is a spanning subgraph of G in a form of a tree.

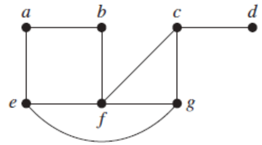
This means that $T = (V_T, E_T)$ is a spanning tree of $G = (V_G, E_G)$ if T is a tree and $V_T = V_G$. A spanning tree of a graph can be obtained by removing circuit on the graph.

Original graph:

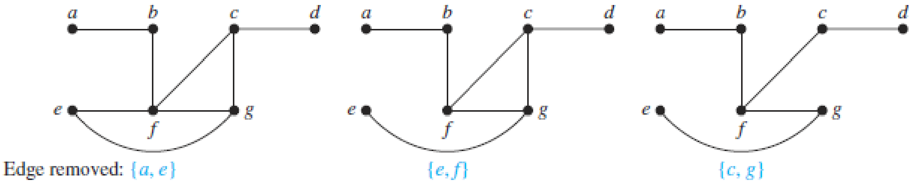


Tree construction:

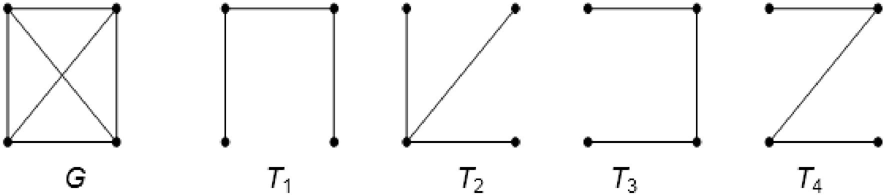
Original graph:



Tree construction:



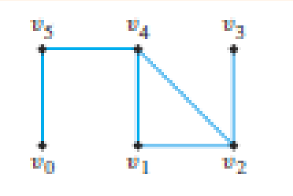
A graph can have more than one spanning tree. Some of spanning trees of K_4 are as follows.



Exercise 1: Finding All Spanning Trees

Exercise

Find all spanning trees of the following graph.

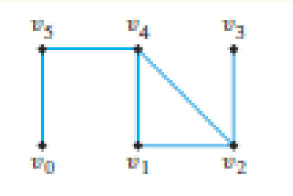


Solution: We have:

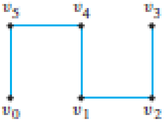
Exercise 1: Finding All Spanning Trees

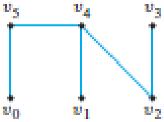
Exercise

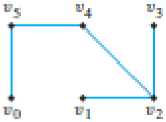
Find all spanning trees of the following graph.



Solution: We have:







Spanning Tree Properties of a Graph

- Every connected graph has at least one spanning tree.
- A disconnected graph with k components has at least k components of spanning tree called spanning forest.

Minimum Spanning Tree

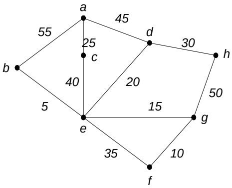
Minimum Spanning Tree

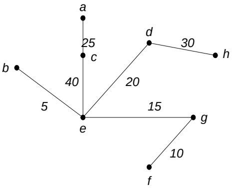
A connected weighted graph may have more than one spanning tree. A spanning tree with minimum weight is called as a minimum spanning tree.

Minimum Spanning Tree

Minimum Spanning Tree

A connected weighted graph may have more than one spanning tree. A spanning tree with minimum weight is called as a minimum spanning tree.





To determine a minimum spanning tree of a graph, we can use two algorithms, namely Prim's algorithm and Kruskal's algorithm.

Prim's Algorithm for Minimum Spanning Tree

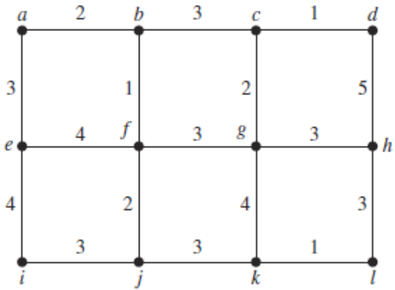
Prim's Algorithm

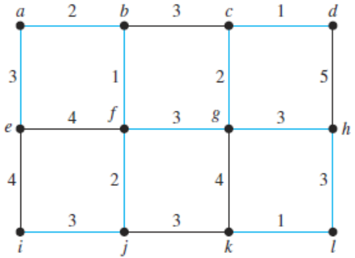
1. `Input:` a connected weighted graph $G = (V_G, E_G)$ and $|V_G| = n$.
2. `Initialization:` $T = (V, E)$ contains **all vertices** on G and $E = \{e\}$ where e has the **minimum** weight.
3. `for` $i := 1$ `to` $n - 2$
4. Choose $e = \{u, v\}$ as an edge that satisfies **all the following criteria**:
5. e has minimum weight `and`
6. e is incident on a vertex in T
7. `if` $T' := (V, E \cup \{e\})$ has no circuit
8. $T := (V, E \cup \{e\})$
9. `else`
10. $T := (V, E)$
11. `Output:` $T = (V, E)$ is a minimum spanning tree.

Prim's Algorithm is one of the examples of *greedy* algorithm, i.e., an algorithm that always take the best choices (edge with the smallest weight) on each of its iteration.

Illustration of Prim's Algorithm

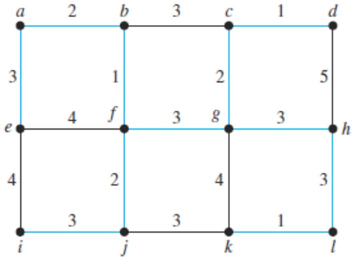
Suppose G is the following graph.





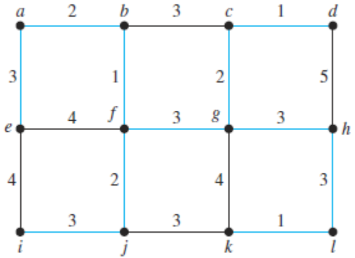
Prim's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge						



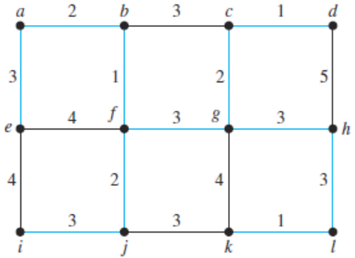
Prim's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{b, f\}$					



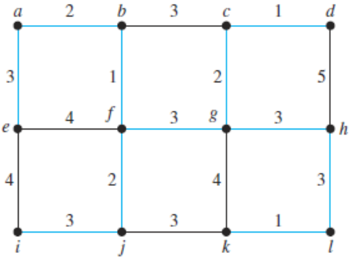
Prim's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{b, f\}$	$\{b, a\}$				



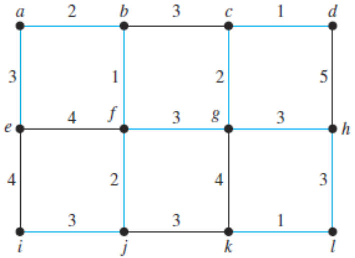
Prim's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$			



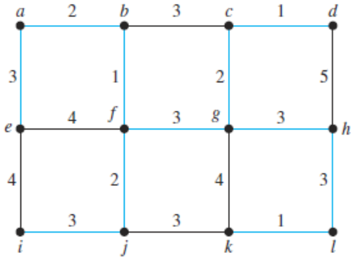
Prim's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$		



Prim's algorithm works as follows:

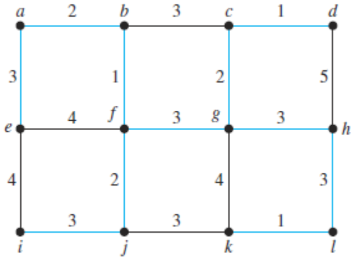
Choice no.-	1	2	3	4	5	6
Edge	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	



Prim's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Weight	1	2	2	3	3	3

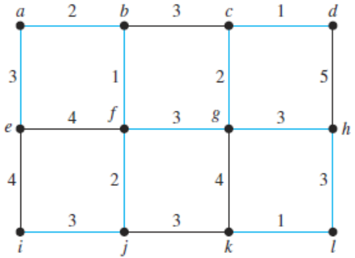
Choice no.-	7	8	9	10	11	Total
Edge						



Prim's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Weight	1	2	2	3	3	3

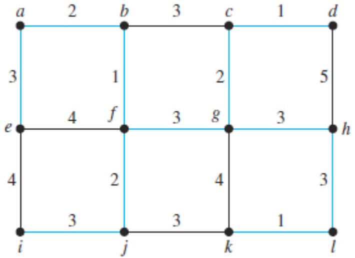
Choice no.-	7	8	9	10	11	Total
Edge	$\{g, c\}$					



Prim's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Weight	1	2	2	3	3	3

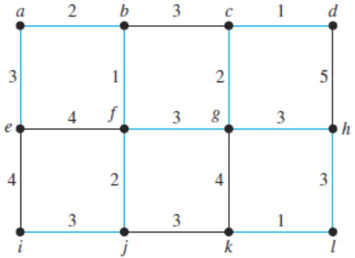
Choice no.-	7	8	9	10	11	Total
Edge	$\{g, c\}$	$\{c, d\}$				



Prim's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Weight	1	2	2	3	3	3

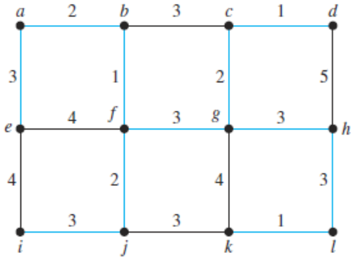
Choice no.-	7	8	9	10	11	Total
Edge	$\{g, c\}$	$\{c, d\}$	$\{g, h\}$			



Prim's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Weight	1	2	2	3	3	3

Choice no.-	7	8	9	10	11	Total
Edge	$\{g, c\}$	$\{c, d\}$	$\{g, h\}$	$\{h, l\}$		



Prim's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{b, f\}$	$\{b, a\}$	$\{f, j\}$	$\{a, e\}$	$\{j, i\}$	$\{f, g\}$
Weight	1	2	2	3	3	3

Choice no.-	7	8	9	10	11	Total
Edge	$\{g, c\}$	$\{c, d\}$	$\{g, h\}$	$\{h, l\}$	$\{l, k\}$	
Weight	2	1	3	3	1	24

Kruskal's Algorithm for Minimum Spanning Tree

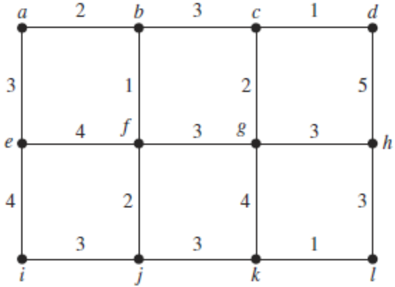
Kruskal's Algorithm

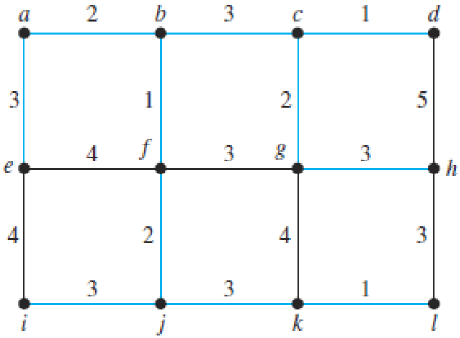
1. `Input:` a graph connected weighted graph $G = (V_G, E_G)$ and $|V_G| = n$.
2. `Initialization:`
3. $\quad T = (V, E)$ with $V = \emptyset$ and $E = \emptyset$.
4. $\quad$ sorts the edges in E_G based on their weight
5. `for` $i := 1$ `to` $n - 1$
6. $\quad$ choose $e = \{u, v\}$ from the sorted E_G
7. $\quad$ `if` $T' = (V, E \cup \{e\})$ contains no circuit
8. $\qquad T = (V, E \cup \{e\})$
9. $\quad$ `else`
10. $\qquad T := (V, E)$
11. `Output:` $T = (V, E)$ is a minimum spanning tree.

Kruskal's algorithm is one example of *greedy* algorithms, i.e., an algorithm that always pick the best choice (edge with the smallest weight) on each of its iteration.

Illustration of Kruskal's Algorithm

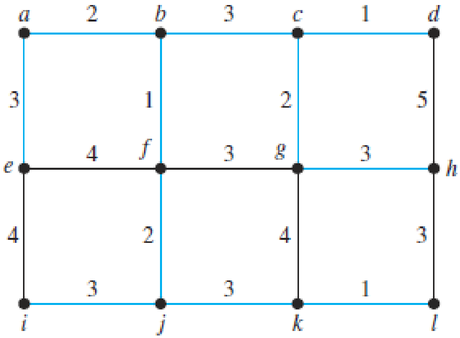
Suppose G is the following graph.





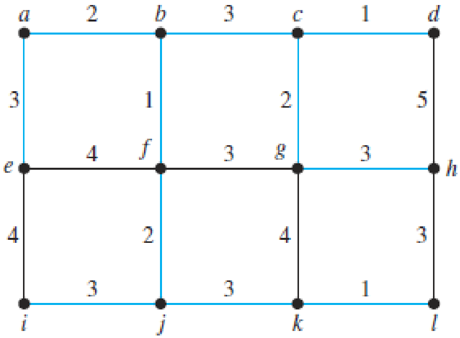
Kruskal's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge						



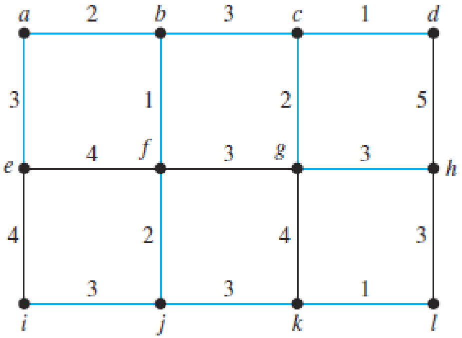
Kruskal's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{c, d\}$					



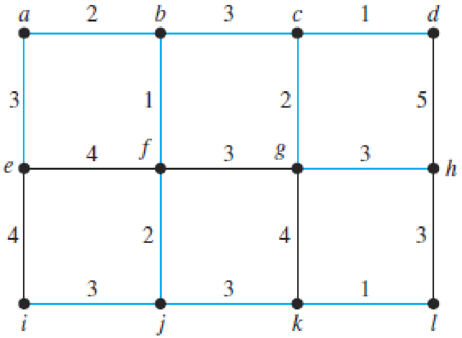
Kruskal's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{c, d\}$	$\{b, f\}$				



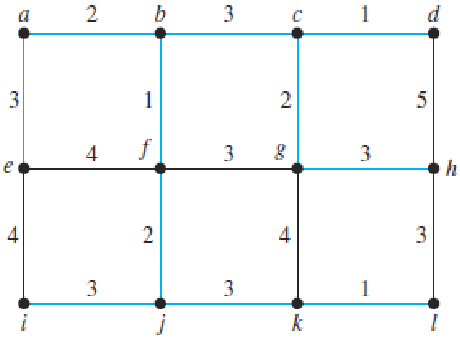
Kruskal's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$			



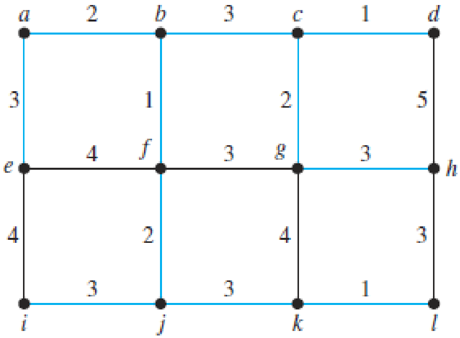
Kruskal's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{c, d\}$	$\{b, f\}$	$\{k, l\}$	$\{a, b\}$		



Kruskal's algorithm works as follows:

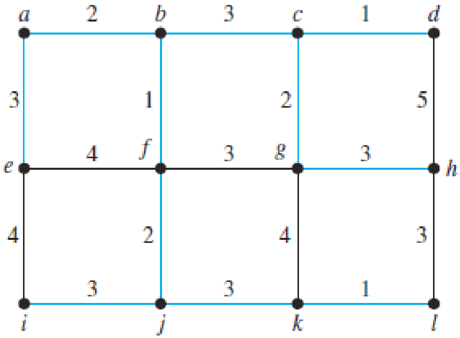
Choice no.-	1	2	3	4	5	6
Edge	$\{c, d\}$	$\{b, f\}$	$\{k, l\}$	$\{a, b\}$	$\{c, g\}$	



Kruskal's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$	$\{c,g\}$	$\{f,j\}$
Weight	1	1	1	2	2	2

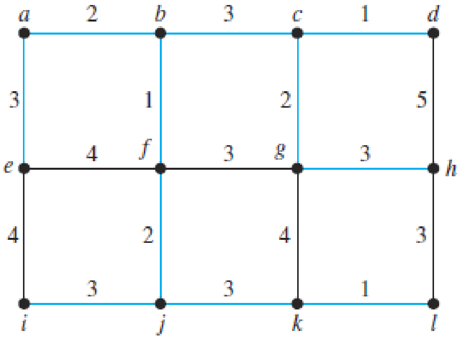
Choice no.-	7	8	9	10	11	Total
Edge						



Kruskal's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$	$\{c,g\}$	$\{f,j\}$
Weight	1	1	1	2	2	2

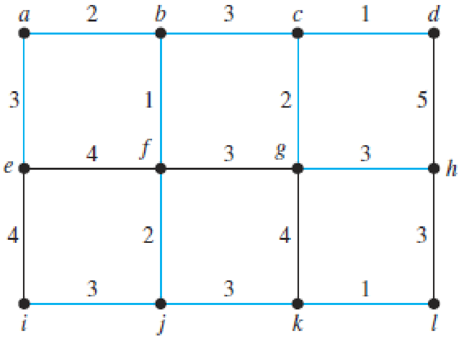
Choice no.-	7	8	9	10	11	Total
Edge	$\{a,e\}$					



Kruskal's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{c, d\}$	$\{b, f\}$	$\{k, l\}$	$\{a, b\}$	$\{c, g\}$	$\{f, j\}$
Weight	1	1	1	2	2	2

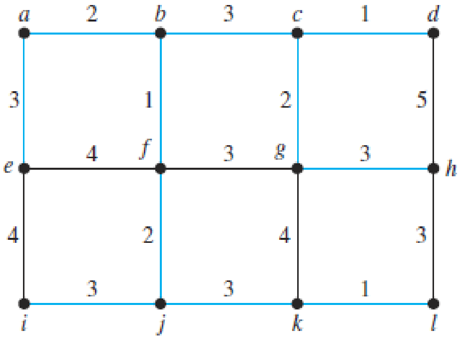
Choice no.-	7	8	9	10	11	Total
Edge	$\{a, e\}$	$\{b, c\}$				



Kruskal's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$	$\{c,g\}$	$\{f,j\}$
Weight	1	1	1	2	2	2

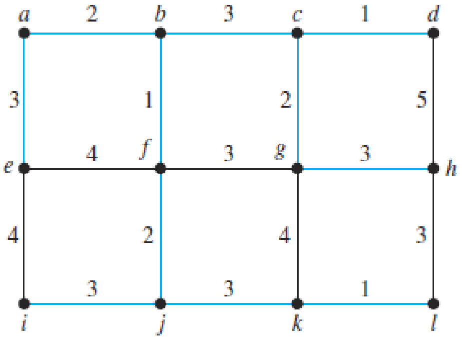
Choice no.-	7	8	9	10	11	Total
Edge	$\{a,e\}$	$\{b,c\}$	$\{g,h\}$			



Kruskal's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{c,d\}$	$\{b,f\}$	$\{k,l\}$	$\{a,b\}$	$\{c,g\}$	$\{f,j\}$
Weight	1	1	1	2	2	2

Choice no.-	7	8	9	10	11	Total
Edge	$\{a,e\}$	$\{b,c\}$	$\{g,h\}$	$\{i,j\}$		



Kruskal's algorithm works as follows:

Choice no.-	1	2	3	4	5	6
Edge	$\{c, d\}$	$\{b, f\}$	$\{k, l\}$	$\{a, b\}$	$\{c, g\}$	$\{f, j\}$
Weight	1	1	1	2	2	2

Choice no.-	7	8	9	10	11	Total
Edge	$\{a, e\}$	$\{b, c\}$	$\{g, h\}$	$\{i, j\}$	$\{j, k\}$	
Weight	3	3	3	3	3	24

Contents

Tree Traversal

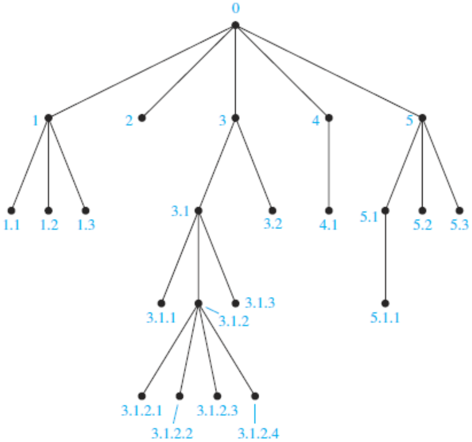
We usually use rooted tree to store information. Therefore, we need a method to visit each vertex on the tree. The visiting process of each vertex is called tree traversal.

Universal Address System

A tree can be used to store information with universal address system which is labeling for each vertex on a rooted tree. Labeling can be done recursively as follows:

- Root is labeled with 0, then if on level 1 there are k children, then each child on level 1 is labeled from left to right with $1, 2, \ldots, k$.
- For every vertex v on level t with label A, if v has n children, then children of v are labeled from left to right with $A.1, A.2, \ldots, A.n$.

Example of universal address designation.



Preorder Traversal

Preorder traversal can be explained recursively as follows.

Preorder Traversal

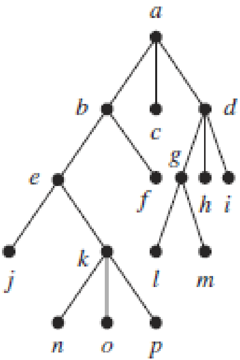
1. `procedure` $preorder\,(T\,(ordered\ root\ tree))$
2. $r :=$ root of T
3. `list` r
4. `for` every child c of r with the order from left to right
5. $\quad T\,(c) :=$ subtree with root c
6. $\quad preorder\,(T\,(c))$

Intuitively, preorder traversal works with the following procedure:

1. visit root,
2. visit left subtree, and
3. visit right subtree.

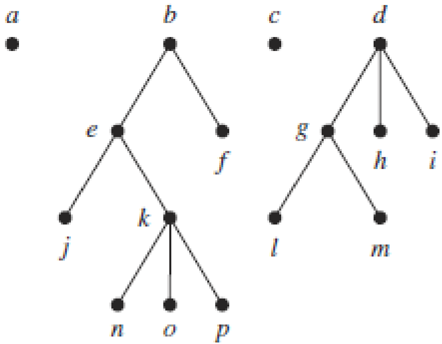
Example of Preorder Traversal

Suppose the tree whose vertices will be ordered is:

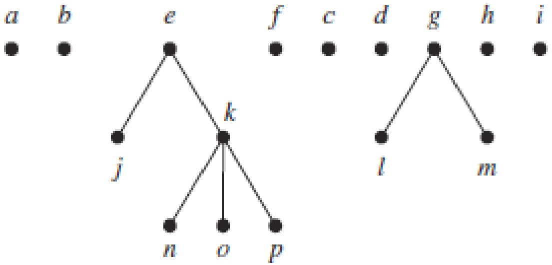


Preorder traversal: Visit root, visit subtrees left to right

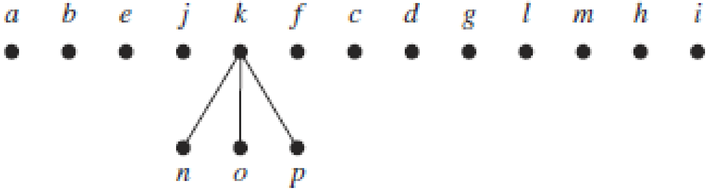
First iteration



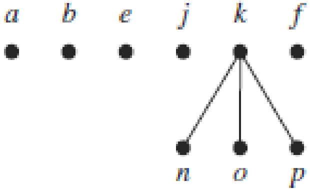
Second iteration



Third iteration



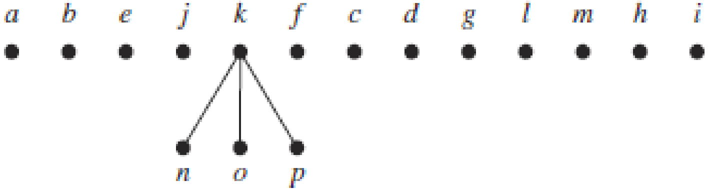
Third iteration



Fourth iteration

So the order of the vertices with preorder traversal is

Third iteration



Fourth iteration

So the order of the vertices with preorder traversal is
$a, b, e, j, k, n, o, p, f, c, d, g, l, m, h, i$.

Postorder Traversal

Postorder traversal can be explained recursively as follows.

Postorder Traversal

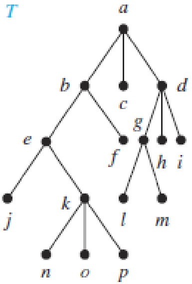
1. `procedure` $postorder\,(T\,(ordered\ root\ tree))$
2. $r :=$ root of T
3. `for` every child c of r with the order from left to right
4. $\quad T\,(c) :=$ subtree with root c
5. $\quad postorder\,(T\,(c))$
6. `list` r

Intuitively, postorder traversal works with the following procedure:

1. visit subtree from left to right, and
2. visit root.

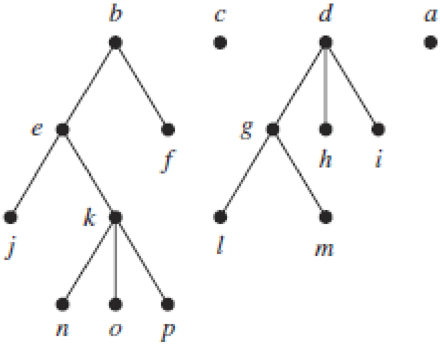
Example of Postorder Traversal

Suppose the tree whose vertices will be ordered is:

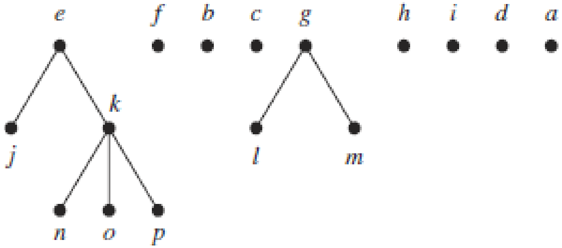


Postorder traversal: Visit subtrees left to right; visit root

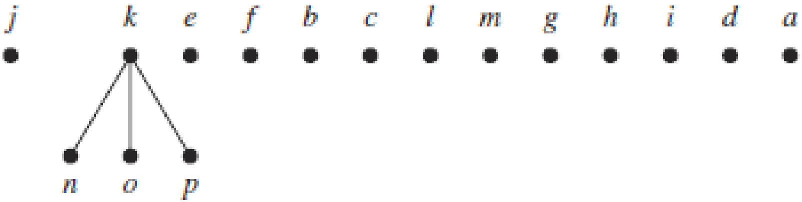
First iteration



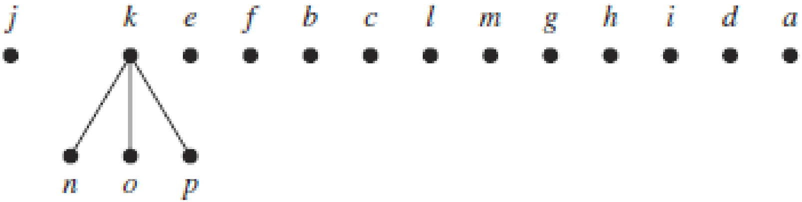
Second iteration



Third iteration



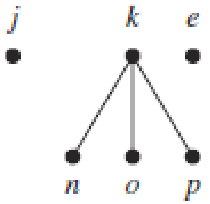
Third iteration



Fourth iteration

So the order of vertices with postorder traversal is

Third iteration



Fourth iteration

So the order of vertices with postorder traversal is
$j, n, o, p, k, e, f, b, c, l, m, g, h, i, d, a$.

Inorder Traversal

Inorder traversal can be explained recursively as follows.

Inorder Traversal

procedure $inorder\,(T\,(ordered\ root\ tree))$

$r :=$ root of T

if r is a leaf **then list** r

else

  $\ell :=$ first child of r from left to right

  $T\,(\ell) :=$ subtree with root ℓ

  $inorder\,(T\,(\ell))$

  list r

  for every child of c from r except ℓ from left to right

    $T\,(c) :=$ subtree with root c

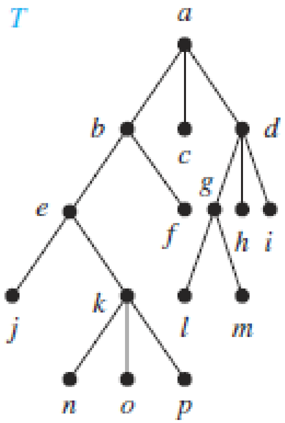
    $inorder\,(T\,(c))$

Intuitively, inorder traversal works with the following procedure:

1. visit the leftmost subtree,
2. visit root, and
3. visit subtree from left to right.

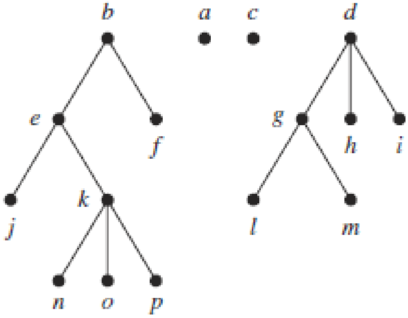
Example of Inorder Traversal

Suppose the tree whose vertices will be ordered is:

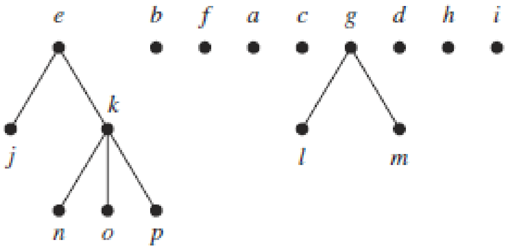


Inorder traversal: Visit leftmost subtree, visit root, visit other subtrees left to right

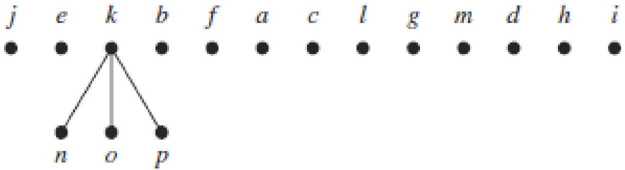
First iteration



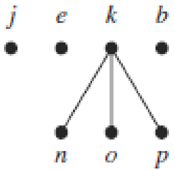
Second iteration



Third iteration



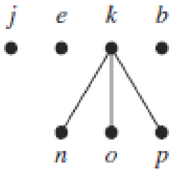
Third iteration



Fourth iteration

So the order of vertices with inorder traversal is

Third iteration



Fourth iteration

So the order of vertices with inorder traversal is
$j, e, n, k, o, p, b, f, a, c, l, g, m, d, h, i$.

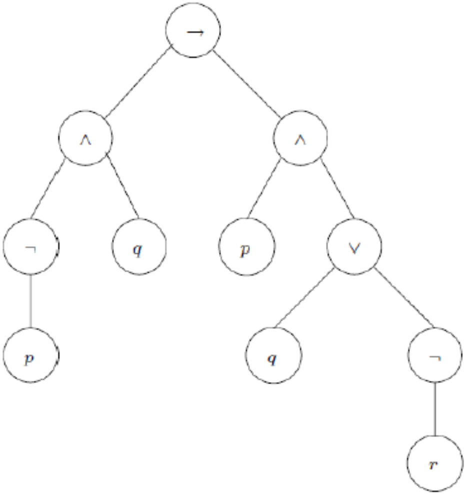
Contents

Parse Tree

In Mathematical Logic-A course, we have learned the parse tree for logical formula. For example, the parse tree for the propositional formula $(\neg p \wedge q) \rightarrow (p \wedge (q \vee \neg r))$ is

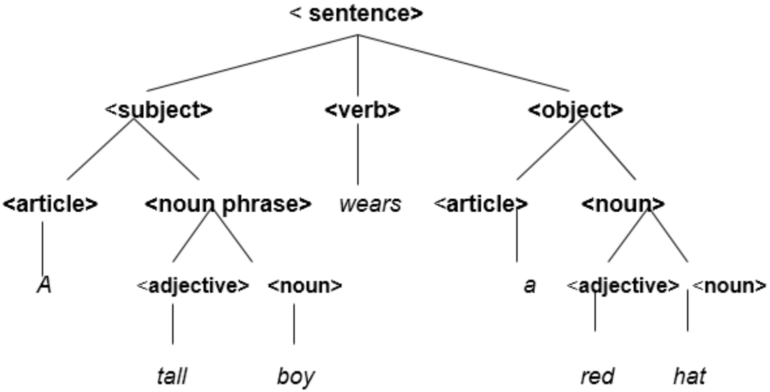
Parse Tree

In Mathematical Logic-A course, we have learned the parse tree for logical formula. For example, the parse tree for the propositional formula $(\neg p \land q) \to (p \land (q \lor \neg r))$ is



Parse tree can be used to parse mathematical expression as well as sentences in particular language. This is one of the foundation of natural language processing. For example, a sentence "*a tall boy wears a red hat*" can be parsed as follows:

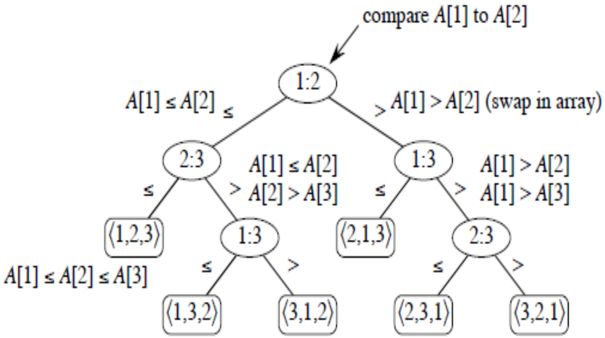
Parse tree can be used to parse mathematical expression as well as sentences in particular language. This is one of the foundation of natural language processing. For example, a sentence "*a tall boy wears a red hat*" can be parsed as follows:



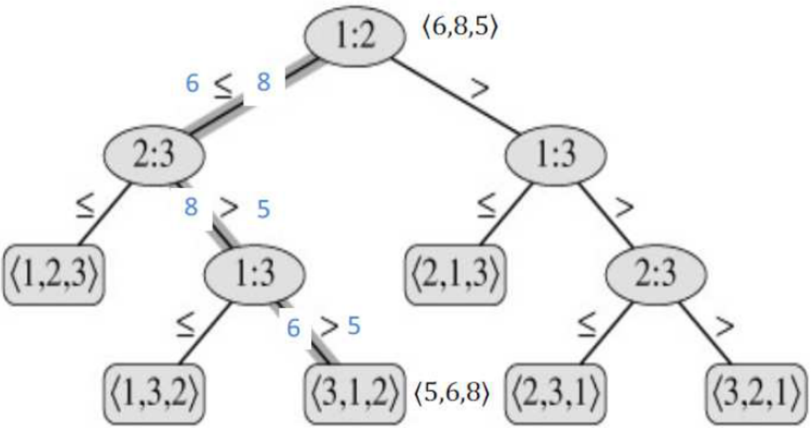
Contents

Decision Tree

A decision tree is a tree that describes the way to take decision on a particular algorithm. Suppose we have an array $A = \langle A[1], A[2], A[3] \rangle$. The algorithm to order the array elements in ascending order can be described in the following tree.



For example, the illustration of array ordering process with of three elements $\langle 6, 8, 5 \rangle$ is



Contents

Mathematical Expression

In high school we have already known a complex mathematical expression that involve more than one operator, such as $2 + 4 \times 5$. Which one of the following value is right:

1. $2 + 4 \times 5 = 30$,
2. $2 + 4 \times 5 = 22$.

When being stored in a computer, a mathematical expression can be represented in at least three ways, namely:

Mathematical Expression

In high school we have already known a complex mathematical expression that involve more than one operator, such as $2 + 4 \times 5$. Which one of the following value is right:

1. $2 + 4 \times 5 = 30$,
2. $2 + 4 \times 5 = 22$.

When being stored in a computer, a mathematical expression can be represented in at least three ways, namely:

- infix notation, (a standard notation that is commonly used in daily life),

Mathematical Expression

In high school we have already known a complex mathematical expression that involve more than one operator, such as $2 + 4 \times 5$. Which one of the following value is right:

1. $2 + 4 \times 5 = 30$,
2. $2 + 4 \times 5 = 22$.

When being stored in a computer, a mathematical expression can be represented in at least three ways, namely:

- infix notation, (a standard notation that is commonly used in daily life),
- prefix notation, using preorder traversal, and

Mathematical Expression

In high school we have already known a complex mathematical expression that involve more than one operator, such as $2 + 4 \times 5$. Which one of the following value is right:

1. $2 + 4 \times 5 = 30$,
2. $2 + 4 \times 5 = 22$.

When being stored in a computer, a mathematical expression can be represented in at least three ways, namely:

- infix notation, (a standard notation that is commonly used in daily life),
- prefix notation, using preorder traversal, and
- postfix notation, using postorder traversal.

Prefix notation is also known as Polish notation and postfix notation is also known as reverse Polish notation.

Precedence of Basic Arithmetic Operator

Precedence of arithmetic operators tells us the priority about which operator has to be executed first on operands.

Precedence of Basic Arithmetic Operator

Precedence of arithmetic operators tells us the priority about which operator has to be executed first on operands.

Precedence table for basic arithmetic operators is as follows.

Operator	Precedence
ˆ (power)	1
$*$ (multiplication)	2
$/$ (division)	3
$+$ (addition)	4
$-$ (subtraction)	5

Precedence of Basic Arithmetic Operator

Precedence of arithmetic operators tells us the priority about which operator has to be executed first on operands.

Precedence table for basic arithmetic operators is as follows.

Operator	Precedence
ˆ (power)	1
$*$ (multiplication)	2
/ (division)	3
$+$ (addition)	4
$-$ (subtraction)	5

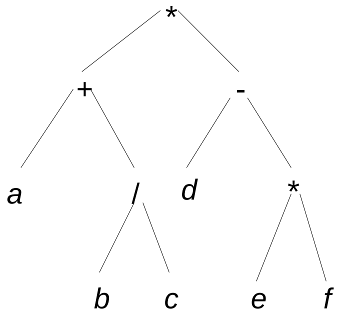
As we learn in high school, we can use brackets "(" and ")" to clarify which operation will be executed first.

Example

Suppose we have a mathematical expression $(a + b/c) * (d - e * f)$. The parse tree for this expression is as follows:

Example

Suppose we have a mathematical expression $(a + b/c) * (d - e * f)$. The parse tree for this expression is as follows:

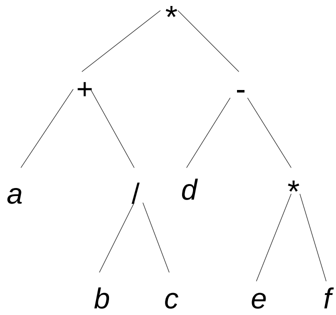


The expression $(a + b/c) * (d - e * f)$ is represented in infix notation. We can obtain prefix notation and postfix notation of this expression using preorder and postorder traversal, respectively, so we have:

- prefix notation:

Example

Suppose we have a mathematical expression $(a + b/c) * (d - e * f)$. The parse tree for this expression is as follows:

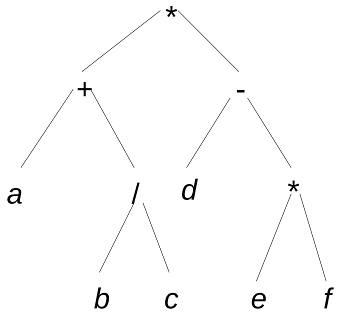


The expression $(a + b/c) * (d - e * f)$ is represented in infix notation. We can obtain prefix notation and postfix notation of this expression using preorder and postorder traversal, respectively, so we have:

- prefix notation: $* + a/bc - d * ef$ (the result of preorder traversal from the parse tree);
- postfix notation:

Example

Suppose we have a mathematical expression $(a + b/c) * (d - e * f)$. The parse tree for this expression is as follows:



The expression $(a + b/c) * (d - e * f)$ is represented in infix notation. We can obtain prefix notation and postfix notation of this expression using preorder and postorder traversal, respectively, so we have:

- prefix notation: $* + a/bc - d * ef$ (the result of preorder traversal from the parse tree);
- postfix notation: $abc/ + def * -*$ (the result of postorder traversal from the

Advantages of Prefix and Postfix Notation

Although it is quite hard to be read by human, prefix and postfix notation have an advantage, i.e., both notations do not need brackets to avoid ambiguity. Therefore, both notation are commonly used in compiler design and development.